图文 103 第三个场景驱动：Broker是如何把自己注册到NameServer去的?

191 人次阅读 2020-02-24 18:00:00

详情 评论

第三个场景驱动：Broker是如何把自己注册到NameServer去的?

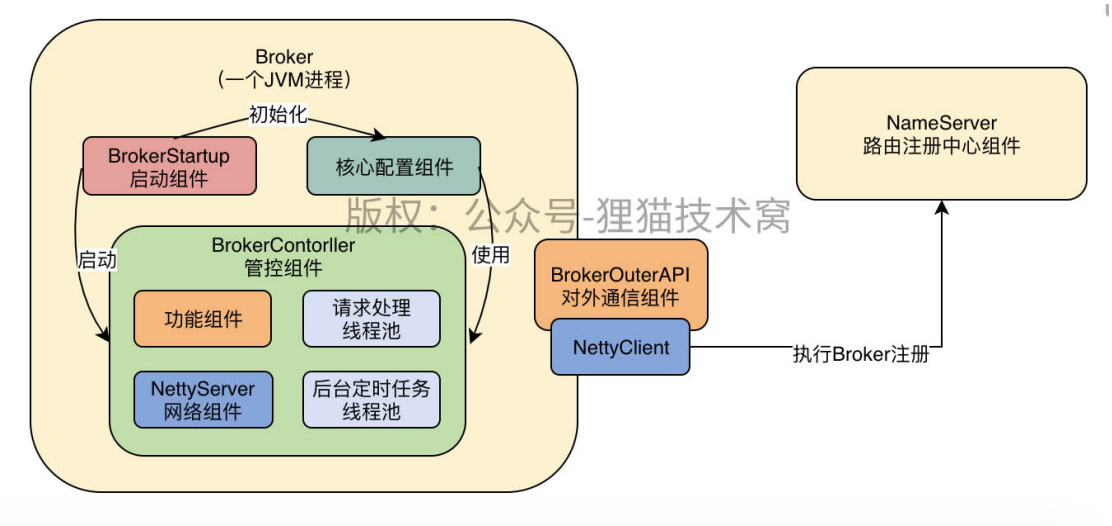
继《从零开始带你成为JVM实战高手》后，阿里资深技术专家携新作再度出山，重磅推荐：

（点击下方蓝字试听）

《从零开始带你成为MySQL实战优化高手》

1、Broker将自己注册到NameServer的入口

上回我们讲到了BrokerController启动的过程，其实他本质就是启动了Netty服务器去接收网络请求，然后启动了一堆核心功能组件，启动了一些处理请求的线程池，启动了一些执行定时调度任务的后台线程，如下图所示，我们回顾一下。



当然，最为关键的一点，就是他执行了将自己注册到NameServer的一个过程，我们看一下这个注册自己到NameServer的源码入口，下面这行代码就是在BrokerController.start()方法中

```
BrokerController.this.registerBrokerAll(true, false, brokerConfig.isForceRegister());
```

因此如果我们要继续研究RocketMQ源码的话，当然应该场景驱动来研究，之前已经研究完了NameServer和Broker两个核心系统的启动场景，现在来研究第三个场景，就是Broker往NameServer进行注册的场景。

因为只有完成了注册，NameServer才能知道集群里有哪些Broker，然后Producer和Consumer才能找NameServer去拉取路由数据，他们才知道集群里有哪些Broker，才能去跟Broker进行通信！

2、进入registerBrokerAll()方法去初步看一看

接着我们就进入到registerBrokerAll()方法中初步的去看一看，大家看下面的源码片段，就是registerBrokerAll()方法的源码，我都写了详细的注释了，大家仔细看一下。

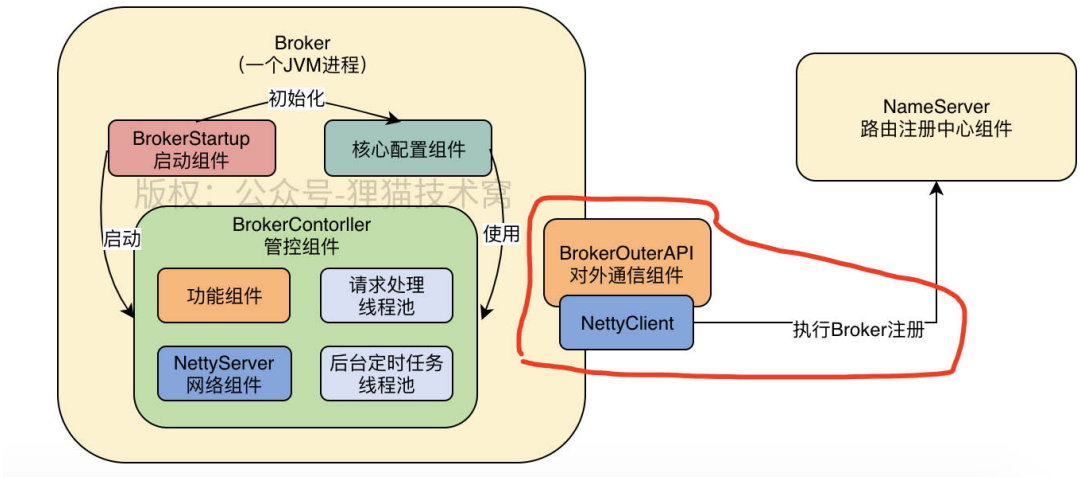
```
public synchronized void registerBrokerAll(

    final boolean checkOrderConfig,
    boolean oneway,
    boolean forceRegister) {

   // 下面这行代码估计很多人看了会一脸懵逼
   // 其实这个没关系，你大致就知道，他就是Topic配置信息相关的东西就行
   // 看源码的时候如果老是深究类似下面这种代码是干什么的，那你绝对会放弃的
   TopicConfigSerializeWrapper topicConfigWrapper =
       this.getTopicConfigManager().
       buildTopicConfigSerializeWrapper();

   // 下面一大段代码，其实你会发现，都是在处理TopicConfig的一些东西
   // 而且里面的一些类名看着稀奇古怪的，可能很多人根本不知道怎么理解他们
   // 其实这个真没关系，你只要脑子里有个印象，这里在搞一些TopicConfig的东西就好
   if (!PermName.isWriteable(this.getBrokerConfig().getBrokerPermission())

           || !PermName.isReadable(this.getBrokerConfig().getBrokerPermission())) {

       ConcurrentHashMap<String, TopicConfig> topicConfigTable =
           new ConcurrentHashMap<String, TopicConfig>();

       for (TopicConfig topicConfig : topicConfigWrapper.getTopicConfigTable().values()) {

           TopicConfig tmp =
               new TopicConfig(topicConfig.getTopicName(),
                               topicConfig.getReadQueueNums(),
                               topicConfig.getWriteQueueNums(),
                               this.brokerConfig.getBrokerPermission());

           topicConfigTable.put(topicConfig.getTopicName(), tmp);
       }
       topicConfigWrapper.setTopicConfigTable(topicConfigTable);
   }

   // 下面这个才比较关键一些，他就是判断了一下，是否要进行注册
   // 如果要进行注册的话，就调用了doRegisterBrokerAll()这个方法，真正去注册
   if (forceRegister || needRegister(this.brokerConfig.getBrokerClusterName(),
                                     this.getBrokerAddr(),
                                     this.brokerConfig.getBrokerName(),
                                     this.brokerConfig.getBrokerId(),
                                     this.brokerConfig.getRegisterBrokerTimeoutMills())) {

       doRegisterBrokerAll(checkOrderConfig, oneway, topicConfigWrapper);
   }
}
```

3、继续探索真正的进行Broker注册的方法

接着我们继续探索真正进行Broker注册的方法，也就是下面的doRegisterBrokerAll()方法，我们进去可以先初步看一下方法的整体情况，我都写了详细的注释，大家也仔细看一看就行。

```java
private void doRegisterBrokerAll(boolean checkOrderConfig,
                                 boolean oneway,
                                 TopicConfigSerializeWrapper topicConfigWrapper) {

    // 说白了，最核心的进行注册的代码就在这里
    // 大家可以看到，这里调用了BrokerOuterAPI去发送请求给NameServer
    // 在这里就完成了Broker的注册，然后获取到了注册的结果
    // 为什么注册结果是个List呢？因为Broker会把自己注册给所有的NameServer!
    List<RegisterBrokerResult> registerBrokerResultList =

        this.brokerOuterAPI.registerBrokerAll(this.brokerConfig.getBrokerClusterName(),
                                              this.getBrokerAddr(),
                                              this.brokerConfig.getBrokerName(),
                                              this.brokerConfig.getBrokerId(),
                                              this.getHAServerAddr(),
                                              topicConfigWrapper,
                                              this.filterServerManager.buildNewFilterServerList(),
                                              oneway,
                                              this.brokerConfig.getRegisterBrokerTimeoutMills(),
                                              this.brokerConfig.isCompressedRegister());

    // 如果说注册结果的数量大于0，那么就在这里对注册结果进行处理
    // 处理的逻辑涉及到了MasterHAServer之类的东西，这里我们就先不用深究了
    if (registerBrokerResultList.size() > 0) {

        RegisterBrokerResult registerBrokerResult = registerBrokerResultList.get(0);

        if (registerBrokerResult != null) {
            if (this.updateMasterHAServerAddrPeriodically && registerBrokerResult.getHaServerAddr() != null) {
                this.messageStore.
                    updateHaMasterAddress(registerBrokerResult.getHaServerAddr());
            }
            this.slaveSynchronize.
                setMasterAddr(registerBrokerResult.getMasterAddr());

            if (checkOrderConfig) {
                this.getTopicConfigManager().
                    updateOrderTopicConfig(registerBrokerResult.getKvTable());
            }
        }
    }
}
```

其实大家看完上面的代码，再看一下下面的图中，我用红圈圈出来的部分，你就会发现，在这里实际上就是通过BrokerOuterAPI去发送网络请求给所有的NameServer，把这个Broker注册了上去。



4、深入到网络请求级别的Broker注册逻辑

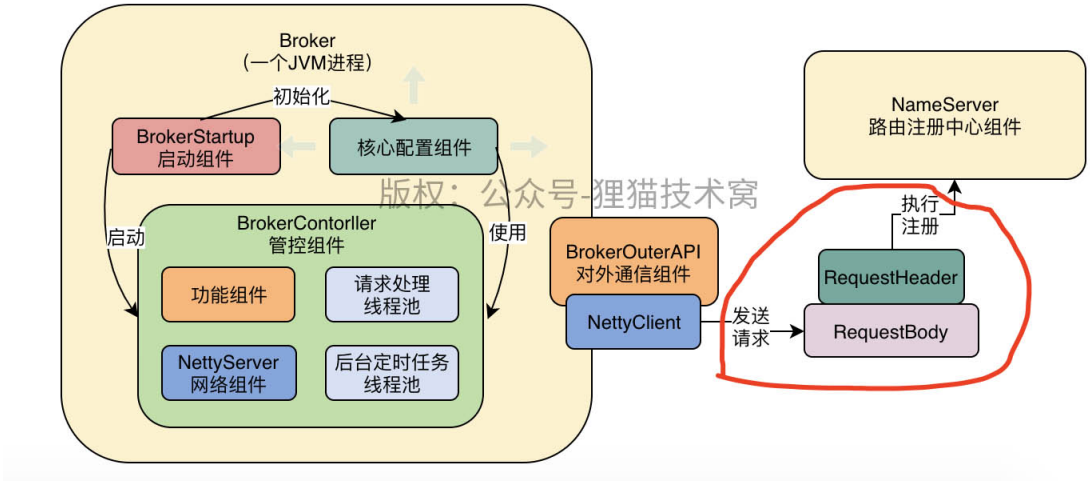
接着我们继续去看BrokerOuterAPI中的registerBrokerAll()方法，就是深入到了网络请求级别的Broker注册了，我给代码写了详细的注释，大家也是仔细看一看。

```
public List<RegisterBrokerResult> registerBrokerAll(

    final String clusterName,
    final String brokerAddr,
    final String brokerName,
    final long brokerId,
    final String haServerAddr,
    final TopicConfigSerializeWrapper topicConfigWrapper,
    final List<String> filterServerList,
    final boolean oneway,
    final int timeoutMills,
    final boolean compressed) {

    // 下面其实就是初始化一个list，用来放向每个NameServer注册的结果的
    final List<RegisterBrokerResult> registerBrokerResultList = Lists.newArrayList();

    // 下面这个list就不用多说了，其实就是NameServer的地址列表
    List<String> nameServerAddressList = this.remotingClient.getNameServerAddressList();

    if (nameServerAddressList != null && nameServerAddressList.size() > 0) {

        // 下面这个很关键，他其实就是在构建注册的网络请求了
        // 首先他有一个请求头，在请求头里加入了很多的信息，比如broker的id和名称
        final RegisterBrokerRequestHeader requestHeader = new RegisterBrokerRequestHeader();

        requestHeader.setBrokerAddr(brokerAddr);
        requestHeader.setBrokerId(brokerId);
        requestHeader.setBrokerName(brokerName);
        requestHeader.setClusterName(clusterName);
        requestHeader.setHaServerAddr(haServerAddr);
        requestHeader.setCompressed(compressed);

        // 下面这个就是请求体了，请求体里就会包含一些配置
        RegisterBrokerBody requestBody = new RegisterBrokerBody();
        requestBody.setTopicConfigSerializeWrapper(topicConfigWrapper);
        requestBody.setFilterServerList(filterServerList);
        final byte[] body = requestBody.encode(compressed);
        final int bodyCrc32 = UtilAll.crc32(body);

        requestHeader.setBodyCrc32(bodyCrc32);

        // 然后在这个地方，他搞了一个CountDownLatch
        // 这个意思就是要注册完全部的NameServer之后才能往下走
        final CountDownLatch countDownLatch =
            new CountDownLatch(nameServerAddressList.size());

        // 在这个地方，就是遍历NameServer地址列表
        // 因为每个NameServer都要发送请求过去进行注册的
        for (final String namesrvAddr : nameServerAddressList) {

            brokerOuterExecutor.execute(new Runnable() {
                @Override
                public void run() {
                    try {
                         // 真正执行注册的地方在这里
                        RegisterBrokerResult result =
                            registerBroker(namesrvAddr,oneway, timeoutMills,requestHeader,body);

                         // 注册完了，注册结果就放到一个list里去
                        if (result != null) {
                            registerBrokerResultList.add(result);
                        }
```

```
                    log.info("register broker[{}]to name server {} OK", brokerId, namesrvAddr);
                } catch (Exception e) {
                    log.warn("registerBroker Exception, {}", namesrvAddr, e);
                } finally {
                     // 注册完了，就会执行CountDownLatch的countDown
                    countDownLatch.countDown();
                }
            }
        });
    }
    // 如果有人对CountDownLatch是什么都不知道？那建议你上网搜一下资料
    // 简单来说，他在这里会等待所有的NameServer都注册完毕了，才会继续往下走
    try {
        countDownLatch.await(timeoutMills, TimeUnit.MILLISECONDS);
    } catch (InterruptedException e) {
    }
}
return registerBrokerResultList;
}
```

上面这段代码里，大家最主要的，是要提取出来RequestHeader和RequestBody两个概念，就是通过请求头和请求体构成了一个请求，然后会通过底层的NettyClient把这个请求发送到NameServer去进行注册

我们看下图，我加入了这个概念。



5、今日源码作业

今天给大家布置一个源码小作业，就是希望大家能够自己在Intellij IDEA里，把今天给大家分析的Broker注册的初步的一些流程都自己看一下，尝试跟我一样，去从乱七八糟的源码里提取出来最重要和关键的一些概念。

比如你应该注意到的是Broker注册的时候，最为关键的BrokerOuterAPI这个组件，然后注意到他里面是对每个NameServer都执行了注册，包括他还构造了RequestHeader和RequestBody组成的请求去进行注册。

如果大家有什么分析源码的心得，可以在评论区里发出来。

End

狸猫技术窝精品专栏及课程推荐：

- 《从零开始带你成为JVM实战高手》
- 《21天互联网Java进阶面试训练营》（分布式篇）